O I P E JC88
SEP 1 3 2004
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Hoa Thu Tran et al.	§	Group Art Unit:	2126
		§		
Serial No.:	09/587,302	§		
		§	Examiner:	Charles E. Anya
Filed:	June 5, 2000	§		
		§		
For:	Controlling Software Components in a Multi-Node Processing System	§	Atty. Dkt. No.:	9172 (NCR.0011US)

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION OF HOA THU TRAN AND MATTHEW DICKEY UNDER 37 C.F.R. § 1.131

We, Hoa Thu Tran and Matthew Dickey, state as follows:

1. We are the inventors of the above-referenced patent application.

2. The document attached as Exhibit A is a copy of relevant portions of the High-Level Software Design Specification created by employees of NCR Corporation (the assignee of the present application) for NCR's TOR 2.0 product. The document attached as Exhibit A is dated on or around October 2, 1998.

3. The subject matter claimed by the present application is described in the document attached as Exhibit A. *See* Exhibit A, pp. 7-17.

4. The software described in the High-Level Software Design Specification was implemented in source code, a portion of which is attached as Exhibit B.

5. The source code of Exhibit B was written in 1998, with completion of the source code of Exhibit B occurring around the December 1998 time period, or earlier. *See* Exhibit B, p. 1 (copyright date in 1998).

6. The source code of Exhibit B was tested around the December 1998 time period. *See* Exhibit C, Feature Unit Test Plan and Test Report for TOR DBMS Startup, dated Jan. 6, 1999. The test schedule indicated test execution occurred between December 3, 1998 and December 18, 1998. Exhibit C, p. 1. The source code was loaded in a database system that substantially duplicated an environment of actual use. *See, id.*, pp. 1-2 (single node and multi-node testing performed). The test indicated that the startup feature described and claimed in the present application performed according to its intended purpose. *See id.*, pp. 23-27.

7. The attached documents establish conception of the invention occurring at least as early as October 2, 1998, and reduction to practice of the invention occurred at least as early as December 1998.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Date: 9/9/2004

[illegible]
HOA THU TRAN

Date: 9th Sept. 2004

Matthew Dickey
MATTHEW DICKEY

541-0001110A04

NCR

High-Level Software Design Specification

for

TOR 2.0

October 2, 1998

Curt Ellmann, William O'Connell
Editors

Parallel Systems

THIS DOCUMENT AND INFORMATION HEREIN ARE THE PROPERTY OF NCR CORPORATION AND ALL UNAUTHORIZED USE AND REPRODUCTION ARE PROHIBITED

Copyright © 1998 by NCR Corporation, Dayton, Ohio U.S.A.
All Rights Reserved. Printed in U.S.A.
CONFIDENTIAL
Unpublished Property of NCR Corporation

Preface

Revision History

Revision/ Version	Author(s)	Principal Contributors	Date	Comments
A01	Curt Ellmann, William O'Connell		7/31/98	First draft for review
A02	Curt Ellmann, William O'Connell		8/24/98	Second draft for review
A03	Curt Ellmann, William O'Connell		9/9/98	Added new version of ODBC Catalogs functions section supplied by Tim Ieong
A04	Curt Ellmann		10/2/98	Update information about directories used by TOR per TRP feedback.

Reviewers of current revision/version

Revision	Reviewer Name	Function(s) Represented by Reviewer
	Karen Kawate	
	Warren Sterling	
	Keith Sweetnam	

Approver: Diane Wewerka

Audience:
TOR engineering, product planning and system architects as well as all staff delegated for technical revieww of this document. This document is meant for internal NCR personnel only.

Additional Information:
Knowledge of TOR architecture is expected. This document does not describe the architecture of TOR, it describes the high level design of all features being done in Release 2.0.

Table of Contents

1 CHAPTER: INTRODUCTION

1.1 Purpose

The purpose of this document is to describe the high-level designs of all software features being integrated into Teradata Object Relational (TOR) Database System, Release 2.0 (R2). This document does not describe the architecture of TOR. The product description (541-0001100A02) and external specification (541-0001111) provide an overview of the system.

1.2 Scope

Each chapter in this document describes the software architecture for a specific feature being added to R2. This document, in particular, is nothing more than the accumulation of individual high-level design documents that were completed for each feature. They are packaged here as individual chapters for convenience to the reader.

The synopsis of each of the chapters follows:

- Chapter 2, Installation and Configuration, describes installing and configuring TOR.
- Chapter 3, Startup and Shutdown, describes starting up and shuting down TOR.
- Chapter 4, ODBC Driver Design, overviews the ODBC driver implementation.
- Chapter 5, Catalog Design, descibes the catalogs
- Chapter 6, Character Data Type Design, describes the enhances to the character data types
- Chapter 7, Decimal Data Type Design, describes the new decimal data type
- Chapter 8, Mixed Data Type Design, overviews enhances to the typing sys. for mixing types
- Chapter 9, Parameterized Types Design, overviews types parameterized on declaration.
- Chapter 10, Lexical Conformance, feature modified the SQL to conform with the standard
- Chapter 11, NULL Support Design, adds NULL support for column data
- Chapter 12, Dynamic UDF Design, discusses user defined types
- Chapter 13, Security Design, enhanced the security features of TOR
- Chapter 14, Tracing and Event Design, overviews event handling and the tracing facility
- Chapter 15, Histogram ADT, describes histogram statistics
- Chapter 16, Query Scheduler Design, overviews the sceduler re-write feature
- Chapter 15, Resource Usage Design, describes how resource counts will be accumulated
- Chapter 17, Storage Maintenance Utility Design, provides information on the utility
- Chapter 18, Text ADT Design, describes the text ADT changes
- Chapter 19, View and Sub-Query Design, overviews the chgs for handling views/sub-queries
- Chapter 20, User Network Design, is a feature done explicitly for NCR Comm's use.

2 CHAPTER : Installation and Configuration Design

The TOR Installation and Configuration features allow the NT administrator to quickly and simply install TOR on a multi-node NT platform. It installs all of the binaries, data files, demo files, and configuration files needed to run TOR on 1 to 8 nodes. Most of the options have defaults in order to simplify the installation. However, the administrator can override these defaults. Solaris is not supported in this release.

2.1 OVERVIEW

2.1.1 Architecture Context and Synopsis

The Installation Tool uses the NT InstallShield product in order to provide a framework that maintains a "NT Look and Feel". This product provides the GUI, file placement utilities, and uninstall utility for a single node. We will add specialized code to replicate the installation to the other nodes in the system. This tool is NT specific and will not run on Solaris. For those unfamiliar with InstallShield, from their WEB site: "InstallShield 5.1 Professional Edition is the complete solution for creating world-class, bulletproof installation systems for Windows applications. InstallShield 5.1 features an Installation Development Environment (IDE) with integrated visual file layout, time-saving Wizards, command-line build programs and templates for key third-party components such as ODBC and DAO."

The Configuration Tool has a GUI that is written in Java. Some of its support routines are written in C++. This tool will be tested only on an NT platform, but it is believed that it will easily port to Solaris when needed. The Configuration Tool allows the administrator to specify the minimal TOR configuration parameters needed to bring up an operational, multi-node TOR system. It cannot be used to reconfigure an existing system. The end result of the tool is the creation of the ".pds_config" file that can be used by the TOR Starter. The Configuration Tool can be optionally started once TOR Installation is complete. Additionally, the Configuration Tool can optionally start the TOR Starter once the configuration has been defined.

2.1.2 Deviations from Requirements

2.1.2.1 Configuration Tool Requirement (TOR.2.9.1.1)

The Configuration Tool meets the requirement for installing an initial configuration. To expand the system after this initial configuration is up and running is beyond the scope of this tool. This tool can only be used at initial install time and cannot be used to reconfigure an operational TOR system.

2.1.2.2 Easy Install Requirement (TOR.2.22.1.1)

The TOR Installation tool meets or exceeds the entire requirement. One caveat is that at this time there are no plans to support installation via PUT (Parallel Upgrade Tool) for first out. However, as PUT for NT reaches DA0 and its requirements become clearer, this may be re-evaluated.

2.1.2.3 Installation Verification Requirement (TOR.2.22.1.2)

We do not plan to produce any type of Installation Verification Tool in the TOR 2.0 timeframe. However, if errors occur during the Installation / Configuration, the user is be notified immediately and can take appropriate action at that time.

2.1.2.4 Registration Wizard Requirement (TOR.2.22.1.3)

The TOR Installation Tool will enter the appropriate information into the NT Registry on each node it is installed on. This information is defined in the document "NCR NT Packaging Requirements Specification", Rev. E, dated March 27, 1998, Jeff Janvrin, Paul Clements, Server Systems in Columbia.

At this time, there are not plans to support the remote registry to a central support service registry. More detailed information is needed about this registry before scoping and design can commence.

2.1.2.5 Uninstallability Requirement (TOR.22.1.4)

InstallShield provides the mechanism to uninstall a product from a single node. The user invokes this uninstall via the Control Panel. Specialized code will be added to this uninstall procedure to remove TOR from all of the other nodes in the system.

2.1.2.6 Efficient Upgrade Requirement (TOR.22.1.5)

This requirement will not be met for TOR Release 2.0. TOR can only be fully installed. There are no formal "Patch Updates" provided via this tool. To install a bug fix in the field, the user or support person will reinstall TOR with the new version. All binaries will be overwritten. Installation of TOR requires that the database not be up. If a previous version of TOR is detected on the system during installation, no TOR Configuration information nor TOR user data will be modified during this installation. However, a developer or support person still has the option of installing a small fix manually without using this tool.

2.1.2.7 Remote Upgrade Requirement (TOR.22.1.6)

This requirement will not be met for TOR Release 2.0. TOR can only be installed via a CD that must be onsite. However, once the CD is onsite, the install can be done remotely via the normal Remote Support console. A developer or support person still has the option of transmitting and installing a small fix manually without using this tool.

2.2 DECOMPOSITION DESCRIPTION

2.2.1 TOR Installation

2.2.1.1 Server and Client Installation Steps

1. Put TOR CD into Query Coordinator (QC) node. By doing this, the Installation Tool can assume that the default QC node is the node that InstallShield is running on.
2. If CD auto startup is enabled, then the TOR Installation Startup Screen will automatically be displayed on the user's monitor. If it doesn't display automatically, the user will have to execute "setup.exe" on the CD.
3. Validate that the user is running in Administrator Mode.
4. Validate that the user is running on a supported version of NT.
5. Gather system info:
    a) Check if TOR has previously been installed on any of the nodes. If it has, then use previous install parameters as defaults for this install.
    b) Validate that TOR is not up.
6. Display "Welcome Screen" that recommends that all windows applications be closed during the installation.
7. Display TOR license screen.
8. Install TOR User Group.
9. Request user information: Name, Company.
10. Request whether this is a server installation, client installation, or both. The default is both.
11. Request whether a typical or custom installation is desired.
12. If this is a server installation, then request names of all of the nodes to install and validate that there is remote access to them.
13. Display the default installation directory "C:\NCR\TOR" with option to change it. This is the directory to which all folders and files will be installed on each node. This directory will be stored in the Registry.
14. Show the "Start Copying" confirmation screen.
15. InstallShield copies folders and files to the destination directory on the local node.
16. If a Client Installation, install ODBC.
17. If a Server Installation, the Installation Tool then copies the directory structure to the selected, remote nodes.
18. If a Server Installation, TOR services are installed on each selected node.
19. Create Program Items and Control Panel service information as needed.
20. Update the NT registry on each node.

21. Show "Installation Completed" successfully screen.
22. If a Server Installation, request if the user wishes to optionally start the TOR Configuration Tool.

2.2.1.2 Uninstall Steps

1. TOR can be uninstalled by going to any TOR node, opening the Control Panel, and selecting the Add/Remove Program icon. TOR will be a listed feature that can be removed.
2. The InstallShield uninstall script is called and removes TOR from the local node.
3. A script is started on all remote nodes to remove TOR from them.

2.2.1.3 Directory Structure

The directory structure for TOR is fixed under the TOR HOME location. The HOME location defaults to " C:\NCR\TOR" with the option to the user to change it. The TOR HOME directory is saved in the NT Registry. This directory structure will be created if it does not exist. The directories directly under HOME into which TOR binaries and
data files are installed are

- bin
- config
- data
- demo
- lib
- diag
- diag/trace
- tmp

2.2.1.4 Licensing

There are currently no known licensing requirements that for NCR NT Products. Anyone with a TOR CD can install TOR. There are not license keys or serial numbers required.

2.2.1.5 Error Logging

TOR Installation logs events to the Windows NT Event Log. Events include installation successful/unsuccessful, services installed, and installation aborted.

2.2.1.6 Registry Entries

Once the TOR Installation is complete, the following Registry Entries are placed on each node (the values listed for each key are examples):

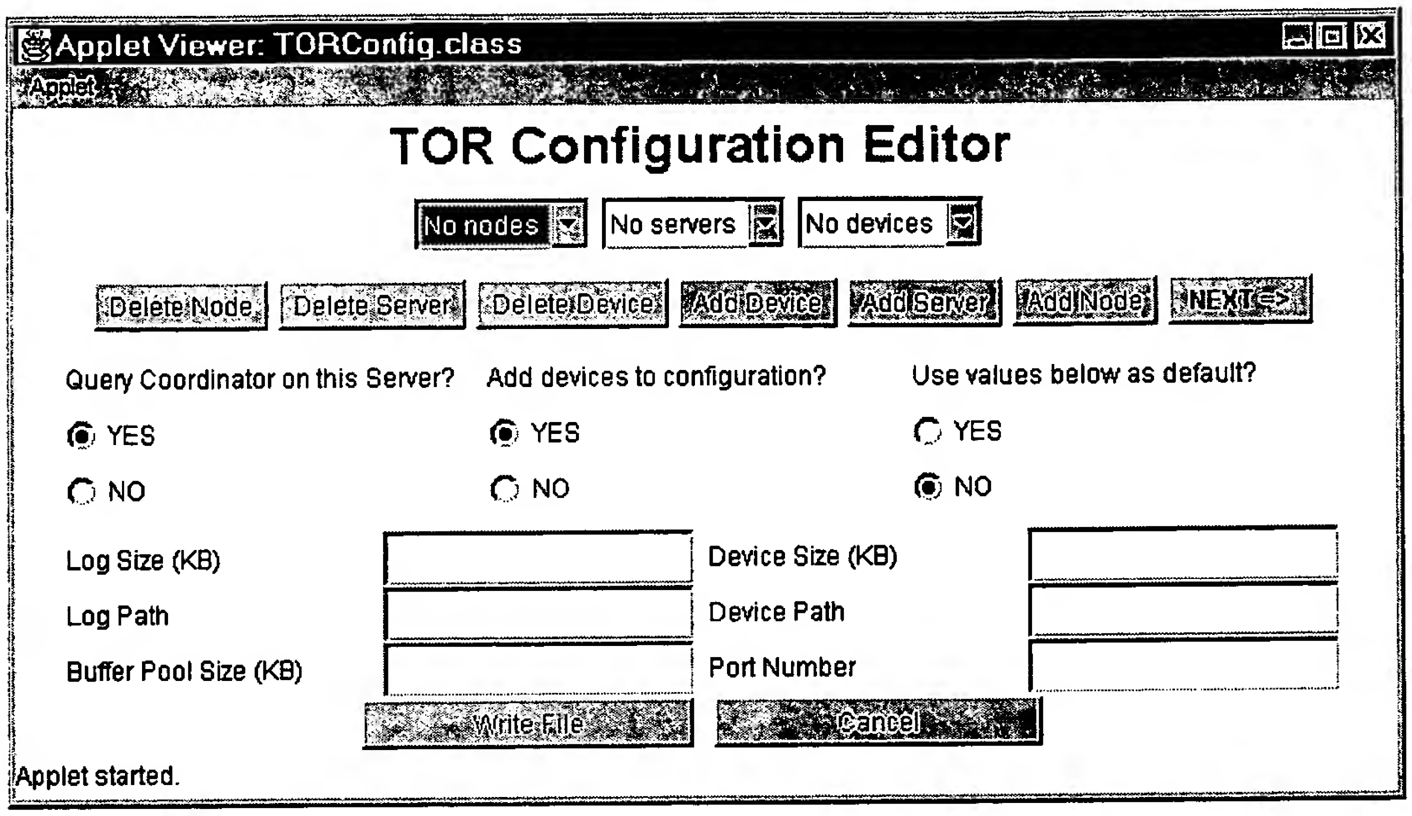
```
HKEY_LOCAL_MACHINE\Software\NCR\ TOR
       CURRENTVERSION="2.00.0000"
       2.00.0000
              TORDIAGDIR="C:\NCR\TOR\diag"
              TORHOME="C:\NCR\TOR"
              TORNODES="SMP-001 SMP-002 SMP-003"
```

where CURRENTVERSION is set to the version number of the last installed TOR package and where each separate version of TOR has its Registry Keys stored under a folder named by the appropriate version number.

2.2.2 TOR Configuration Tool

2.2.2.1 Configuration Steps

1. The TOR Configuration Tool is either startup via the Installation Tool at its completion or manually by the user. This tool is written in Java and will run either via an Internet browser or as an independent Java applet. The tool should be started on the node that will be the Query Coordinator (QC). This allows the tool to provide a default node name for the QC. The end result of the tool is the creating of the file .pds_config.



Query Coordinator on this Server? [✓ YES] NO

Add devices to configuration? [✓ YES] NO

Use values below as default? YES [✓ NO]

Log Size (KB)		Device Size (KB)	
Log Path		Device Path	
Buffer Pool Size (KB)		Port Number	

Applet started.

2. Three pull down menus are displayed at the top of the Configuration Tool that lists all of the defined nodes, servers on that node, and devices on that server. The user can use this menus to quickly switch between nodes/servers/devices or can use the "NEXT" button to advance one record at a time.
3. The Configuration Tool can be used to create a default TOR configuration for a newly installed TOR system. If an existing configuration file exists, the user will be notified to either edit the existing file or to start from scratch and overwrite the existing file.
4. The Configuration Tool automatically gathers a list of existing node names that was passed to it from the previous run of TOR Installation. The user may delete a node configuration by pressing the "Delete Node" button or add a node by pressing the "Add Node" button. The user may not delete the QC node.
5. The QC configuration information is requested via the dialog box where the default QC node is the node that the Configuration Tool is currently running on.
6. As the user moves through the configuration records using the "NEXT" button, he sees that the default number of servers is one per CPU except for the QC node where it is one less than the number of CPU's. The user may delete servers if desired by pressing the "Delete Server" button and add servers by pressing "Add Server".
7. After obtaining the information for the QC node and the user presses "NEXT", the Configuration Tool automatically propagates default values to other nodes and servers by selecting "YES" on the "Use values below as default?" question.
8. The user can then go to each node/server combination to modify the remaining system information. Device information is not required for any server except for the QC. If the user does not wish to specify device information, he can select "NO" on the "Add devices to configuration?" question.
9. When the user has accepted all of the inputted information, and has pressed the "Write File" button, the Configuration Tool writes the .pds_config file to disk and copies it to all other nodes.

2.3 REQUIREMENTS TRACEABILITY

Table 1: Market and Product Requirements for Teradata Object Relational, #541-0000974 A03

Reqmt. Number	Requirement Descriptor	Section List
2.9.1.1	Configuration Tool Requirement	

2.22.1.1	Easy Install Requirement	
2.22.1.2	Installation Verification Requirement	
2.22.1.3	Registration Wizard Requirement	
22.1.4	Uninstallibility Requirement	
22.1.5	Efficient Upgrade Requirement	
22.1.6	Remote Upgrade Requirement	

2.4 GLOSSARY

node – in this document, the term node refers specifically to an SMP box; an MPP system consists of multiple nodes.

QC – the Query Coordinator, which may also be referred to as the master node in some Paradise documents.

server – Data server, also sometimes referred to in some Paradise documents as a "node".

3 CHAPTER: Startup and Shutdown Design

System startup, which is implemented by starter.exe, is a separately executable component of the Teradata Object Relational (TOR) database system. It is used to start Data Servers throughout an SMP or MPP system, and to monitor the Data Servers for possible failures. This document describes how the starter will be adapted to be fully functional in the Release 2.0 Windows NT environment.

The Release 2.0 version of the TOR Starter feature will provide

- multinode startup on MPP systems,
- a degree of intranode and internode monitoring and recovery of Data Servers,
- and file propagation for tracing support.

For Release 2.0 of TOR it will not offer support for node failover.

In the original Paradise package, starter.exe on Windows NT systems did not have the ability to start Data Servers on remote nodes due to a lack of remote shell (rsh). The new version will allow remote startup of Data Servers through use of NT Services, which does not require rsh. A useful side effect of using NT Services is that it allows some monitoring of the status of Data Servers on remote nodes, and a convenient means to automatically restart failed Data Server processes.

Also lacking in the original NT version of Paradise is a means of propagating files throughout an MPP system. This is used in getting consistent trace information to the SMP nodes running TOR. The file propagation capability will be added by using direct addressing of remote drives.

The Market and Product Requirements for Teradata Object Relational document, (#541-0000974, revision A03) [1] lists a number of requirements which this feature helps to satisfy:

3.1.1.1 TOR 2.21.1.2/Ease of Use/2/3/pm, gsc/

The Product shall provide interfaces that are easy to use as exhibited by best in class products.

The startup feature will allow the TOR Database on an MPP system to be started under normal circumstances with no more than five mouse clicks: Start, Settings/Control_Panel, Services, select TOR Starter Service, and Start Service.

3.1.1.2 TOR 2.21.1.10/Common Look and Feel/2/2/pm/

The Product shall provide a common 'look and feel' for all interactions with product provided client tools and utilities.

The point-and-click interface is identical to that use for starting any other NT Service program.

3.1.1.3 TOR 2.22.1.1/Easy Install/1/2/PM/

The Product will provide a procedure that allows for the easy installation and initialization of a system. As far as possible, the install shall be automated and require minimum interaction. A default configuration should be provided in addition to the necessary and automated help required to guide the installer through what may be a significant number of complex options and decisions.

Installation of the service components will be a completely transparent phase of the Installation and Configuration Utilities (see Chapter 2). No additional user interaction is required beyond what is necessary for specifying the SMP nodes and Data Servers.

3.1.1.4 TOR 2.26.1.1/TOR on NT/1/10/pm/

The product shall be delivered using NT as the underlying operating system. This includes both client and server components.

The new version of starter will enable a full-fledged NT offering instead of the original single-node NT implementation.

3.1.1.5 TOR 2.26.1.2/TOR on NCR SMP hw/1/3/pm/

The Product shall operate on a NCR SMP 4300, 4700 or the equivalent hardware platforms available at the Product release timeframe.

The multinode startup capability allows TOR to be started on multinode platforms such as the 4700. Any Intel-based NT system will be able to run the TOR startup feature, including the 4300.

3.1.1.6 TOR 2.26.1.3/TOR on NCR MPP hw/1/2/pm, eng/

The Product shall operate on a NCR MPP platform.

The new version is specifically targeted at fulfilling this requirement. It allows TOR to be started easily and efficiently on NT-based MPP platforms.

The proposed modifications to the starter will have some modest effects on the configuration component, requiring it to set up the NT Services needed. The design also will increase the number of active processes on each SMP node, although without appreciable impact on overall node or system performance.

3.2 OVERVIEW

This section gives an overview of the architecture and its implications in several categories.

3.2.1 Architecture Context and Synopsis

The original Paradise package contains a program, starter [.exe], which is used to start up Query Coordinator and the Data Server processes. It also provides a way to set up the Database configuration, either interactively or through a prepared configuration file. Once the DB is started, the starter program can act as a monitor to ensure the continued correct behavior of the QC and the Data Servers. On Solaris systems, starter.exe can bring up servers on multiple nodes through the use of the remote shell; on Windows NT, however, the remote shell is not always available, and the original Windows NT version of starter.exe lacks the ability to start up Data Servers on remote nodes. Additionally, the monitoring feature on NT is not yet fully functional. In order to provide a multinode NT offering, we need to address these principal shortcomings with a comprehensive approach to database startup. The NT services capability can be used as a means to this end, providing a clean mechanism for starting processes on remote nodes and a way to check on the status of these processes at any time.

3.2.1.1 Background Context

The following is a list of the missing or stubbed-out features in the original NT starter. These are the items which we will remedy for release 2.0 (or release 3.0 in the case of node failover).

1. **No Remote Shell support on NT**: the `start_cmd_exec()` function basically does not work for remote execution under Windows, generating an error message to the effect that

rsh is not yet implemented. Windows creates a command line with the desired program to be executed, sets up process and startup information structures, and calls `CreateProcess()` with them, returning the `pid` of the new process, which means `start_cmd_exec()` will work, but only for local commands. Remote shell can be used on NT if the proper resource kit is available, but it does not provide us with the necessary capabilities to monitor the remote processes which it spawns; the local rsh process cannot be used as a proxy because it often terminates while the remote process is still up, if the connection remains quiescent, which is the expected state of affairs.

2. **Missing Non-Local Data Server Startup**: in `start_master()`, the actual calls to `start_cmd_exec()` are `ifdef`'d out for Windows, without any replacement whatsoever. The copy of the .node and .device files is thus a no-op for windows.
3. **Print Status function**: the `print_status()` function is not available on NT – this function uses the Unix `WIFEXITED` and `WEXITSTATUS` macros, which do not have exact equivalents on NT. They are used to print out error information when something goes wrong spawning a server.
4. **Process Kill mechanism**: the `kill_proc()` function does nothing under Windows. It is used by the Unix ports to kill all the data servers if the Query Coordinator dies or if a Data Server dies and no recovery has been requested on the starter.exe command line (`-r` option).
5. **File Propagation**: File propagation from the starter's SMP node to the other SMP nodes in the system is needed for such things as tunable parameters, tracing flags, and configuration files. Perl scripts are invoked from the starter in order to create the tracing flags. This copying is done by directly accessing drives on remote nodes. Invoking Perl is done using the current `start_local_exec()` function.
6. **Separation of Configuration Code from Starter Code**: Release 2.0 will see a new configuration utility, with the implication that starter will no longer need to perform this function. However, since this function is invoked through explicit use of the '-I' option, we need not actually remove the configuration code from starter.
7. **Monitoring a Multinode NT System**: in order to increase database availability and recoverability, Data Servers, the Query Coordinator, and the starter itself will be monitored for any process failures. For Unix ports of Paradise, this is done by retaining the process id's of any and all child processes of the starter, and waiting on any death-of-child signal. Currently in the Unix version, it appears that if the 'recover' command line option is used, only failures of Data Servers are detected and restarted; otherwise, all the Data Servers are killed and the database is shutdown completely. Under Windows, only the Query Coordinator is monitored (by waiting for it to exit), and no restarts of Data Servers are done. There is no detection of Data Server failures. For the new version of starter on NT, recovery of lost Data Servers will always be attempted.
8. **Error handling and tracing requirements**: In order to conform to TOR requirements more consistent error handling and tracing code is needed. Currently error messages in the starter are simply sent to `cerr` or `cout`. All messages should be sent to the standard trace logs. This problem is confounded by the fact that when the starter is executing code to bring up the database, the standard logs used by the database may not yet be available, so separate provisions may have to be made, such as putting the errors into the NT event log. The starter will also need to invoke Perl scripts for parsing trace flags and must be able to propagate the trace flag information to other SMP nodes. A method will be provided to determine the required filenames for this purpose, which will likely involve reading a registry entry to determine the proper TOR home directory to search.
9. **Internationalization Stubs:** provisions must be made to allow for the eventual internationalization of all messages, prompts, and user-defined strings.

To simplify the startup procedure, the configuration activities will be separated into a new program while the starter functions will be based on the NT Services model, which provides a straightforward means to achieving much of the necessary functionality. A brief description of the NT Services feature follows here; more details are available in the Visual C++ 5.0 help files, under

Platform, SDK & DDK Documentation | Platform SDK | Windows Base Services | Executables | Services.

NT Services is a way to start daemon-like processes automatically or on demand, and control their execution. A key feature is that services can be started on remote nodes without the use of the remote shell facility. This makes it possible for an NT multinode TOR server to be brought up from a single NT node with minimal user input, completely resolving one of our basic multinode issues.

Briefly, an NT service consists of a service program which runs in a loop waiting for input. The input may consist of user requests from a control program, or of system events such as messages or process terminations. Services are run under the auspices of the Service Control Manager (SCM) but can be configured and managed by the user, either programmatically, from a command prompt, or through the Control Panel Services applet.

3.2.1.2 Architectural Synopsis

In the NT environment, starter and the Data Servers (including the Query Coordinator) will be configured as NT services by the new configuration utility. To avoid impacting the Solaris code line, most of the NT-specific implementation will be contained in a separate executable; server.exe will be unaffected. To accomplish this, there will be an additional service process for the starter and for each Data Server. The main tasks of each service process are

- Handling the NT Services protocol
- Spawning either the starter or a Query Coordinator/Data Server process
- Monitoring the spawned process.

To start up TOR, the user starts the TOR System Startup service (via the Control Panel or the 'net start' command). This action will cause NT to kick off the starter service program. After registering with the SCM, the starter service program spawns the regular starter process (starter.exe). Then, starter.exe will initiate server services, one for each configured Data Server in the system.

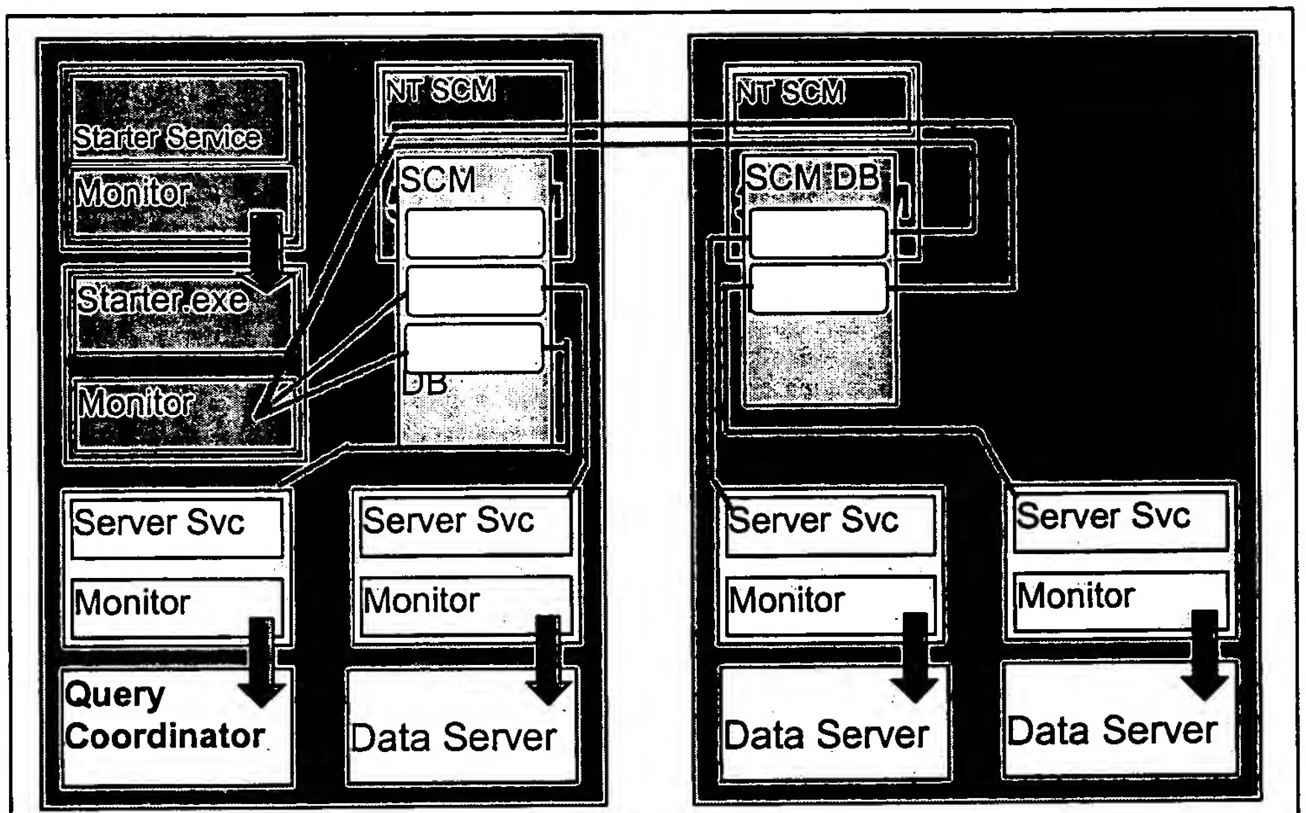


Figure 1: a 2-node system with 2 data servers per node. Red lines indicate monitoring operations. Blue boxes represent NT system components.

Each server service will register with the SCM and spawn one Data Server (the first one of which will be the master, or Query Coordinator). Once the spawning is complete, each service program retires to a monitoring loop in which it checks for possible termination of the Data Server that it spawned (or in the cases of the starter service, it will monitor starter.exe). The diagram (Figure 1) explains this far more clearly than words ever could. The following list gives a brief outline of the steps involved in starting all the Data Servers:

- The database administrator starts the TOR Starter Service via the Services applet or an MS-DOS command line.
- The starter service program spawns starter.exe, passing to it all the parameters which the starter service program received.
- The starter.exe program opens its ocomm and remembers its own endpoint.
- The starter parses the parameters it has received from the starter service program.
- The starter reads the .pds_config file to create a list of Data Servers to start and monitor.
- The starter propagates any necessary configuration files to the node where the Query Coordinator will run (which will ordinarily be on the same node as the starter, making this step very simple).
- The starter collects any configuration information which needs to be passed as parameters to the Query Coordinator.
- The starter starts the server service program which will spawn the Query Coordinator, passing along all the parameters needed by the QC.
- The first server service program comes up and spawns the QC, passing through all the parameters from the starter. Once the QC comes up, the first server service program enters a wait state, waiting for the QC process to terminate.
- The starter waits for the QC to register with it.
- The QC registers with the starter.
- The starter enters a loop which starts all the server services for the regular Data Servers.

- Each new server service program spawns its associated Data Server, passing through all the parameters sent by the starter. Once the new Data Server has started, the server service program waits for the Data Server to terminate.
- The starter waits for the Data Servers to register with it.
- The Data Servers register with the starter, and receive endpoint information about the Query Coordinator.
- Once all the expected registrations have been received, the starter enters its monitoring loop, periodically polling the service programs for any server process terminations which they may have encountered.
- The TOR database is now fully operational.

3.2.2 Usage or Functionality Constraints

Note that the release 2.0 implementation is NT specific. The original implementation does work in the Solaris environment and does not require changes.

Process/server monitoring is limited to the death of a process. Detecting a hung process is beyond the scope of this release. Once TOR detects that a process has died, server recovery will be attempted in place, i.e., restarting the server on the same SMP node. There is no support for node failover in this release. Also, recovery attempts will stop after 3 unsuccessful tries.

3.3 DECOMPOSITION DESCRIPTION

This chapter explains each of the components required to complete the starter services feature. The overall performance impact should be quite modest. The main effect will be that the number of processes will be doubled. The new service processes will spend nearly all their time in a wait state, consuming minimal resources.

3.4 Service Configuration Function

The Service Configuration Program will be a relatively simple C++ function which supports the following operations:

- Installing a TOR starter/server service
- Removing a TOR service

The service configuration function is normally invoked by the configuration utility after a new configuration has been determined.

3.4.1 Initial Service Installation

Installation will begin by creating the service with sufficient access permissions to the SCM services database to ensure that all the other functions can be deployed. This can be done by setting the initial dwDesiredAccess parameter on the CreateService call to have GENERIC_READ, GENERIC_WRITE and GENERIC_EXECUTE permissions.

The dwServiceType parameter should be set to SERVICE_WIN32_OWN_PROCESS so that the service program will run independently of other services. The dwStartType parameter will be set to SERVICE_DEMAND_START.
The level of error control to be used if the service fails to start properly is specified by the dwErrorControl parameter. We suggest that a level of SERVICE_ERROR_NORMAL level, which will merely log the failure and report it to the user via a message box pop-up.

At present no dependencies on other services have been identified for TOR.

Each service will be installed so that it will be started using the TOR built-in account and password. This ensures that each service will have sufficient SCM privileges on remote nodes.

Any errors occurring during the CreateService process will be propagated back to the user, unless they can be rectified and retried by the Service Configuration Program.

3.4.2 Service Removal

Uninstalling the Paradise Starter Service is a simple matter of opening a line of communication to the SCM via the OpenService API call giving the service name used when the service was originally created. The OpenService call provides a handle to be used when calling the DeleteService API, which will remove the service entirely from the SCM Services database.

3.5 Initiating Starter.exe

At least two methods will be available for initiating the starter.exe program. From an MS-DOS prompt, the "net start" command may be invoked which will get the service going. Any needed parameters for the starter can be included on the command line.

The starter service program may also be started through the Services applet on the Control Panel, which provides for parameters to be passed to the service; these in turn will be passed to the starter when it is executed by the service program.

3.5.1 Starter Parameters

The starter service program only needs parameters in order to be able to pass the correct parameters to starter.exe as it is executed. The original parameters which needed to be set for the Starter program can be synopsized by the getopt() option string from starter.cpp:

```
hiflc:V:p:PF:aD:qbrs:m:BSM
```

They are each explained in somewhat greater detail in the following table.

Table 1: TOR Starter.exe Command-Line Parameters

Opt	Meaning	Option Argument	Used in Rel 2
a	Set debug_auto_run to TRUE.	n/a	yes
b	Do not log.	n/a	yes
c	Get the TOR configuration info from the specified file.	Path name	no
f	Format the disks	n/a	yes
h,?	help - print usage	n/a	yes
i	Initialize	n/a	no
l	Avoid using rsh (remote shell).	n/a	no

m	Run specified Data Server in Manual mode.	Data Server name	yes
p	Port number to use	Integer	no
q	run configuration step only	n/a	no
r	Try to recover failed Data Servers	n/a	no
s	Specify resolution scaleup	Integer	yes
B	Run as a background process.	n/a	no
D	Specify a Data Server to debug	Data Server name	yes
F	Get port number from a specified file	path name	no
M	turn on Monitoring	n/a	yes
P	Get the port number from the default port file.	n/a	no
S	Skip death notifications.	n/a	no
V	Verbosity level of messages	Integer	yes

For release 2.0 of TOR, not all of these parameters are deemed necessary The Port number to be used will always be found in the default port number file in the TOR home directory, rendering /p, /F and /P extraneous. Also, since the configuration is now performed as a separate step, the /i option will never be sent by the starter service program to starter.exe Failed Data Servers will always attempt recovery, making the /r option unnecessary. The last column of Table 1 shows which parameters will be viable for release 2.0.

3.6 Starter Service Program

The function of the starter service program is to handle the NT Services protocol and initiate the execution of the starter.exe program, which it will accomplish by doing a `CreateProcess()` system call. It can be started through the Control Panel's Services applet, or with the "net start" command from an MS-DOS prompt. Any arguments for starter.exe need to be passed to the starter service program, which will simply pass them through without verification.

Once the starter service program has started the starter.exe process, it will retire to a monitoring loop waiting for the starter to terminate. It can also take further input from the starter control program in the form of simple status queries and a shutdown command.

Status queries will basically be limited to saying whether the starter.exe process is still running or not.

A shutdown command will cause the starter service program to exit, without affecting the operation of the starter.exe process, other than that it will no longer be monitored locally.

3.7 Starter.exe – Service Control

The principal change required to the starter.exe program for release 2.0 is that instead of calling `createProcess()` itself in order to kickoff server.exe, it will start the appropriate server service program based on the server/node ID. The starter.exe program will be act as a service control program for the Query Coordinator and the Data Servers. The current Solaris functionality will be left intact via use of `ifdef`'s.

A high-level view of the starter's control flow:
1. Process its arguments
2. Read server configuration from the .pds_config file
3. Copy trace file to all SMP nodes

4. Start the Query Coordinator and the Data Servers by starting their corresponding NT services.
5. Once all the Data Servers have been started, the starter is ready to enter its monitoring loop. It will periodically check each svsp via a status request. Any failed Data Servers detected by the monitoring operation can be restarted by restarting the svsp, which will in turn recreate the Data Server. If the Data Server which has failed happens to be the Query Coordinator, the starter will have the server service programs for all remaining Data Servers exit via a `kill_proc()` function.

3.8 Server Service Program (svsp)

The server_service_program (svsp) is started by starter.exe as an NT Service once for each Data Server. Its function to do a local `CreateProcess()` function to start the Data Server going. When the svsp is started, it receives all the startup parameters for the Data Server from the starter.exe program. After kicking off the Data Server process, the svsp goes into a wait for either the termination of its child Data Server process, or a request from the starter.exe program, which is processed by a request handler. When its child Data Server dies, either normally or abnormally, the svsp process kills itself. The starter will detect that the svsp has stopped and restart it if necessary.

The svsp will respond to a status request through the SCM from the starter with the current condition of its child Data Server (or Query Coordinator).

The svsp may also receive a stop request from the starter, if the svsp's child Data Server process has terminated and the starter wishes to restart it. In this case the svsp will do any necessary cleanup and exit.

The svsp may receive a kill request from the starter, indicating that the svsp should perform a `kill_proc()` function on its child Data Server, most likely because the Query Coordinator has died. The mechanism for the `kill_proc()` function will probably have to be `TerminateProcess()`, which does not do a complete job of cleaning up – DLL's which were in use by the Data Server will not be notified that the Data Server is exiting.

3.9

REQUIREMENTS TRACEABILITY

This appendix lists the requirement numbers and descriptors from the Requirements Specification(s) that relate to this specification. For each requirement, numbers of the section(s) in this specification which address them are listed in the Section List column.

Table 1: **Market and Product Requirements for Teradata Object Relational document, (#541-0000974, revision A03)**

Reqmt. Number	Requirement Descriptor	Section List
2.21.1.2	**3.9.1.1 Ease of Use/2/3/pm, gsc/**	3.2
2.21.1.3	**3.9.1.2 Remote interaction/1/2/pm, gsc/**	3.2.1.1, 3.2.1.2
2.21.1.10	**3.9.1.3 Common Look and Feel/2/2/pm/**	3.3.1, 3.3.1.1, 3.3.2
2.22.1.1	**3.9.1.4 Easy Install/1/2/PM/**	3.3.1, 3.3.1.1
2.26.1.1	**3.9.1.5 TOR on NT/1/10/pm/**	All
2.26.1.2	**3.9.1.6 TOR on NCR SMP hw/1/3/pm/**	3.2.1, 3.2.1.1
2.26.1.3	**3.9.1.7 TOR on NCR MPP hw/1/2/pm, eng/**	All

3.10 DESIGN JUSTIFICATION

This design was selected because it provided a straightforward path to getting TOR on NT up in a multi-node environment, with functionality equivalent to the Solaris version of TOR, within the given Release 2.0 schedule constraints. Solutions such as using Remote Shell from the Resource Kit failed to offer a means of monitoring the Data Servers for failures, although it does allow starting the Data Servers on a multinode system. Adapting TOR and the Teradata PDE layer to one another would have introduced unacceptably large schedule uncertainties.
Using NT Services allows the needed multi-node functionality to be implemented with very little impact on the Data Servers and Query Coordinator.

3.11 UNRESOLVED ISSUES

The entire area of SMP node failover has not been addressed by this design.

3.12 GLOSSARY

DS – Data Server, also sometimes referred to in other Paradise documentation as a 'node'. In TOR terminology, node has another meaning, however.

Data Server – a process, running on an NT-based SMP node, which handles TOR database access on behalf of client processes. Multiple Data Servers may be run on a single SMP node.

node – in this document, the term node refers specifically to an SMP box; an MPP system consists of multiple nodes.
node failover – a mechanism whereby relevant processes and disk resources from a failed SMP node are transferred to another surviving SMP node and restarted, transparently to the user.

QC – the Query Coordinator is a distinguished instance of a data server that handles all direct interaction with client processes. The query coordinator may also be referred to as the master node in some Paradise documents.

SCM – the Services Control Manager, a portion of Windows NT devoted to handling services (daemon processes/threads and drivers).

service configuration program – a generic name for a program which opens the **SCM** Services database and registers a **service program** in it.

service control program – a generic name for a program which activates or shuts down a registered **service program**.

service program – a generic name for a program on Windows NT which is registered with the **SCM**, and which functions similarly to a [Unix] daemon process.

SMP node – (= Symmetric Multi-Processor node) a hardware box containing multiple CPU's using shared memory. An SMP node is the basic unit of parallelism in a Massively Parallel Processing system.

starter service program (*also called* **starter service**) – a **service program** which basically exec's the starter.exe program and then goes into a loop monitoring starter.exe for termination.

server service program (*also called* **server service**) – a **service program** which starts a Data Server process and then monitors that child process.

```
/*<std-header>

          Copyright 1998, NCR Corporation.  All Rights Reserved.

                             NCR Confidential.


 $Id: starter.cpp,v 1.52 1998/06/16 00:59:05 jiebing Exp $


  -- do not edit anything above this line --    </std-header>*/

///////////////////////////////////////////////////////////////////////
//
// starter.cpp :  configure, start and monitor all the paradise servers
//
///////////////////////////////////////////////////////////////////////

#define MANAGER

#include <os_fcntl.h>
#include <w_workaround.h>
#include <w_signal.h>
#include <stdlib.h>

#ifdef TOR_NT
#include <w_windows.h>
#include <direct.h>

#else
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#endif

#ifndef TOR_NT
#include <unistd.h>
#endif

#include <string.h>
#include <stdio.h>

#ifdef TOR_NT
#include <time.h>
#else
#include <sys/time.h>
#endif


#ifndef TOR_NT
#include <sys/wait.h>
#endif

#include <new.h>

#ifdef TOR_NT
#include <io.h>
#else
#include <sys/uio.h>
#endif
```

```
#include <sys/stat.h>
#include <errno.h>

#include <iostream.h>
#include <w_strstream.h>
#include <fstream.h>
#include <asciiStream.h>
#include <config_info.h>
#include <KRtypes.h>


#include <scomm_internal.h>

#include <piextern.h>
#include "torconfsvc_gen.h"
#include "regread.h"
#include "svr_svc.h"
#include "su_svc.h"
#include "TORConfig.h"

#if defined(Sparc) && !defined(SOLARIS2)
#include <vfork.h>
#endif

#if defined(Mips) && !defined(Irix)
extern "C" int vfork();
#endif

#if defined(Mips) && defined(Irix)
#define vfork fork
#endif

#if defined(AIX32) && !defined(AIX41)
#define vfork   fork
#endif

#if defined(AIX32) || defined(AIX41)
static void hack_aix_signals();
#endif

// pid_map_t class moved up here to allow references to it before main
().
class pid_map_t
{
  public:
    w_link_t    link;           // used in hash table
    pid_t       pid;
    int         node;

    pid_map_t() { pid = 0; node = -1; }
    ~pid_map_t() {}
};

// This hack keeps anyone from having to once and for all decide where
// messages should go.
enum displayMode_t {COMMAND_LINE = 1, ERROR_OUTPUT = 2, TRACE_FILE = 4,
                              MESSAGE_BOX = 8, DEBUG_FILE = 16 };


// Groddy global variables so we don't have to pass them absolutely
```

```
everyplace.
// Smart routines will put in their own newlines if they're clever.  Or,
since
// this is for Windows NT, they may want to put in CR-LF or whatever the
// magic dingus is for this context - \00a\00d??  Maybe the
compiler/preprocessor
// will be charitable and substitute the appropriate items in place of a
\n.
char displayBuffer[1024];

int displayMode = DEBUG_FILE;
int traceMode   = DEBUG_FILE;

FILE* DebugFile;

// Despite Glenn's protests to the contrary, this would probably work
// a whole lot better as a macro.  But the use of 'ec' inside it might
// break some of the perl scripts.
//
// By making use of the fact that displayMode is a global, we don't need
// to pass it as a parameter (as it was in the first pass at coding this
dog).
// Plus, we can make a separate routine for trace messages and purchase
// another increment of functionality at modest cost.
//


void
displayMessage(char *Buffer = displayBuffer)

{
/*
if (displayMode & COMMAND_LINE) {
            cout << Buffer;
      }
      if (displayMode & ERROR_OUTPUT) {
            cerr << Buffer;
      }

      if (displayMode & MESSAGE_BOX) {
            MessageBox(NULL, Buffer,
                  NULL, MB_OK | MB_DEFAULT_DESKTOP_ONLY);
      }
*/

      if (displayMode & DEBUG_FILE) {
            fprintf(DebugFile, "%s", Buffer);
            fflush(DebugFile);
      }
}

// Default use of displayBuffer simplifies each reference to this
function
// a bit more, but would it be better as the final argument?  For real
use
// of the trace functions, ec will be necessary.  I tried making a dummy
ec
// to serve as a default ec parameter to this function and to
displayMessage,
// but it runs up against a) the lack of a comparison operator for
EventContext
```

```
// objects (so I can't tell if ec == dummyec), or b) after initializing
the
// dummyec with a phony, unlikely filename ("Shmutz", to be exact), a
strcmp
// of ec.fileName with "Shmutz" won't compile because, of course,
fileName
// is a private data element of the EventContext object.  So, we require
an
// ec parameter for traces, and require its absence for displays.  Feh.


// Might could tack on a newline to end of Buffer if there isn't one
// there already.  E.g. if (Buffer[strlen(Buffer)] != '\n') strcat
(Buffer,
// "\n");  Same probably goes for displayMessage().
// We end up using that technique a fair bit.

// Required to have a constant string for consumption by the
// messaging facility's preprocessing scripts.  *Everything*
// interesting is in displayBuffer.  displayBuffer now has to be used by
this
// routine, or else use the bare tracing calls.

// For final version, reinsert call to ec.TraceCheck(category).  Removed
for
// preliminary debugging.  Also removed appends of 'endl' for cout/cerr
// output, since all displayBuffer sprintf's have '\n' at the end
already.
#define traceMessage( category ) { \
if (traceMode & TRACE_FILE)    ec.Trace(category, "(Torstart): %s",
displayBuffer); \
if (traceMode & DEBUG_FILE) {  fprintf(DebugFile, "%s", displayBuffer);
\
      fflush(DebugFile); } } \


//
// maximum size of a command
//
static const int MAX_LINE_SIZE = 2000;

//
// debug print of arguments to start the server
//
void print_args(const char* args[], int nargs, const char* ns = "Args:
")
{
      TOR_EVENT_CONTEXT("::print_args");

    int i;
    sprintf(displayBuffer, "(Torstart) %s\n    ", ns);
    for (i=0; i<nargs; i++) {
            strcat(displayBuffer, args[i]);
            // Run 'em off at 6 per line.  This may or may not be a
            // good plan, but it's probably no worse that the earlier
            // version which just spewed them all to a single line.
            if ((i%4) == 3) strcat(displayBuffer, "\n    ");
            else strcat(displayBuffer, "  ");
     }
    strcat(displayBuffer, "\n");
      traceMessage (EventContext::basic);
```

```
}


//
// Help instructions
// It might could make more sense to sequester this information in,
like,
// a File and simply dump it out when requested.  At least for windows,
// where the formatting is 2nd generation C technology.
//
#ifdef TOR_NT
static const char* instructions =
"\nUsages:\n\n"
"1. To configure TOR from scratch (first time):\n\n"
"\tUse the TOR COnfiguration utility, which can be invoked automatically
\n"
"\tfrom the Install Shield when the TOR package installation is
complete.\n"
"\tThe configuration utility uses a [largely] self-explanatory GUI to\n"
"\tcollect the parameters and information needed to construct the
tor_config\n"
"\tfile, such as devicesm catalog device, buffer pool size, log size,\n"
"\tdirectories, etc.\n\n"
"\tThe configuration utility will ask if you wish to install services.
You\n"
"\tshould choose 'yes'.  After the installation of the NT services is\n"
"\tcomplete, you will be asked if you wish to start the TOR database.\n"
"\tIf you select 'yes', you will also be asked if you wish to have the
\n"
"\tdisk devices reformatted.  For initial installations, answer 'yes'.\n
\n"
"2. To Restart a configured TOR DBMS:\n\n"
"\ta) From the Control Panel 'Services' applet:\n"
"\t\tSelect the service named 'TOR Starter Service' by clicking on it.
\n"
"\t\tIn the parameter box, add any parameters you need to run the\n"
"\t\ttorstart.exe program (ordinarily there should be none, but you\n"
"\t\tmay wish to reformat your disks or specify a verbosity level;\n"
"\t\tplease see the previous section on available switches for\n"
"\t\ttorstart.exe for more information.\n"
"\t\tFinally, click the 'Start' button.\n\n"
"\tb) From an MS-DOS command line prompt:\n"
"\t\tUse the 'torsc' utility to start the TOR Starter Service\n"
"\t\tby using the following command syntax:\n"
"\t\t\ttorsc [\\\\QCnode] torstart [parameters]\n"
"\t\twhere the optional UNC parameter gives the name of the SMP\n"
"\t\tnode where the Query Coordinator will run; this parameter is\n"
"\t\tnot needed if torsc will be run on that SMP node.  The optional\n"
"\t\t'parameters' parameter can be replaced by any parameters that\n"
"\t\tcan be sent to torstart.exe; please see the previous section on\n"
"\t\tavailable switches for torstart.exe for more information.\n"
"\t\tTorsc is found in TORHOME/bin.\n\n"
"\tc) Using torstart.exe directly:\n"
"\t\tThis technique is not recommended for general use, since\n"
"\t\tit bypasses the NT service monitoring, reducing the database's\n"
"\t\tability to recover from various types of failures.\n"
"\t\tFrom an MS-DOS prompt (or from within a script, bash, etc.),\n"
"\t\tuse the following syntax:\n"
"\t\t\ttorstart [parameters]\n"
"\t\twhere the parameters are any of the parameters mentioned in\n"
"\t\tthe section on runtime switches for torstart.exe.  Note that\n"
```

```
"\t\tyou should only run this command from the same SMP node where\n"
"\t\tthe Query Coordinator will run.  Torstart.exe can be found in\n"
"\t\tTORHOME/bin.\n\n"
"3. Using the -m option for manually starting Data Servers:\n\n"
"\tThe option switch '-m <server_number>' allows a Data Server\n"
"\tto be started manually.  Torstart.exe will print out a [long]\n"
"\tlist of parameters with which server.exe should be executed.\n"
"\tFor your convenience in using this option, however, a script\n"
"\tnamed 'suDS<server_number>' (or 'suQC<server_number>', if you\n"
"\thappen to wish to start the Query Coordinator manually) is\n"
"\tcreated on the SMP node where the Data Server will run.\n"
"\tSimply execute the script to start the Data Server.\n\n"
"\tIf for any reason you wish to run the Data Server without\n"
"\tits associated NT service monitor, edit the script to change\n"
"\t'torsc TORsrvr<server_number>' to be 'server' on the first\n"
"\tcommand line of the script.  You may also edit the script to\n"
"\tallow startup of the Data Server from another node by inserting\n"
"\tthe UNC node name of the SMP node where the data server is to\n"
"\nbe run into the script just after 'torsc '.\n\n"
"\tPlease note that if you run a Data Server manually without its\n"
"\tNT service, the tordbabort utility will not be able shut down\n"
"\tthat Data Server.\n\n"
"4. To run the Storage Maintenance Utility (SMU):\n\n"
"\tSimply execute 'torstart -u' from a command line prompt.\n";

#else
static const char* instructions = // See what a *clever* compiler can
do:
"
Usages:

1. Configure paradise server from scratch (first time):

   Use 'starter -i', this will prompt you to type in a set of
   configuration information for each node, including all the
   devices, catalog device, buffer pool size, log size, directories
   etc, ...

   The starter program will then save your configuration option into
   a default file named as 'tor_config' and start everything up by
   formatting all the disks.

   If you want the options to be saved in a specified file instead of
   the default file, use 'starter -i -c <filename>'.

2. Restart the server (don't format the disks):

   Use 'starter' (read config info form tor_config) or
   'starter -c <filename>' (read config info from filename).

3. Initialize server using existing configuration file:

   Use 'torstart -f' (read config info form tor_config) or
   'torstart -f -c <filename>' (read config info from filename).

4. The old options like '-p <portnum>' and '-P' still work. Also
   '-V <verboselevel>' specifies verbose level to all master/locals.

5. When '-D <node_id>' is specified, the starter program will skip the
   invocation of rsh command on that node, instead, it starts a xterm
   and runs a gdb using all the arguments of the command.  If you want
   to debug more than one node, use multiple '-D <node_id>' arguments.
   If you use the '-a' flag with '-D <node_id>' , a gdb 'run' command
   will be issued, causing the debugged node(s) to proceed.  The '-a'
flag
   is applied to all the debugged nodes uniformly.

6. When '-q' is specified together with '-i', then the servers will not
be
   started, only configuration is done.

7. 'starter -i' has a useful feature of automatically setting up rest of
   the server's information (except for hostname) based on the first
server.
   This can be used to configure paradise in a symmetric environment
like
   cops, where most of the options (the device names, binary paths) are
   mirrored on all the data servers.

8. '-l' flag will avoid rsh if possible (if everything is on same
machine).

9. The format of the configuration file is very straightforward, you
   may chose to edit that file directly. See server/src/.pds_confg.tmpl
   for examples.

10. '-b' flag will turn off logging on all nodes.

11. '-m <server_id>' flag will allow manual start of one local, it
prints out
   all the arguments to start that local.

12. '-B' flag will start all servers in background mode.

13. '-S' flag will suppress death notification (useful during debugging,
    so that death notification does not kill the server you are
debugging).

Note:
   For now, if server (master/local) failed in the middle of startup
phase
   due to configuration errors (invalid device, port not available,
...),
   the starter hangs. I created a utility
/p/paradise/util/scripts/mclean
   (based on pclean) to be used to cleanup all the rsh processes.  I
will
   work on a long term solution later. ( -- JBY)
";
#endif

//
// print help on command line
//
void print_usage(ostream& err_stream, char* prog_name)
{
   err_stream << "Usage: " << prog_name << " -ufVmhs" <<endl;
   err_stream << "Switches:" << endl
      << "\t-u: run as Storage Maintenance Utility (SMU)" << endl  //
krw-smu-001
      << "\t-f: format all the disks" << endl
      << "\t-V: <verbose_level> verbose printing for servers" << endl
      << "\t-m: <Data Server #> start Data server manually" << endl
    << "\t-h: print help messages and exit" << endl
      << "\t-s: <res_scaleup> set resolution scaleup to <res_scaleup>"
<< endl
      << "Obsolete manual mode switches:" << endl
    << "\t-i: initialize all configuration parameters" << endl
      << "\t\t(Use TOR configuration utility to set configuration
parameters.)" << endl
    << "\t-q: only prints the configuration without starting servers"
<< endl
      << "\t\t(Only worked with -i option.)" << endl
    << "\t-c: <filename> configuration file" << endl
      << "\t\t(Configuration information is always taken from " << endl
      << "\t\tTORHOME/config/tor_config.)" << endl
    << "\t-p: <portnum> port number for master to listen for clients"
<< endl
    << "\t-P: bind to a free port and write to .paradise.port.number"
<< endl
    << "\t-F: <filename> bind to a free port and write to <filename>"
<< endl
      << "\t\t(Port # is always taken from TORHOME/config/tor_port.)"
<< endl
    << "\t-D: <node_id> start gdb in xterm for specified node" << endl
    << "\t-a: after starting gdb, automatically \'run\' the process" <<
endl
      << "\t\t(Debug options only available on Solaris systems.)" <<
endl
    << "\t-b: turn off logging" << endl
      << "\t\t(Logging is now always enabled.)" << endl
    << "\t-B: start servers in background" << endl
      << "\t\t(Servers are always started in background, unless
started" << endl
      << "\t\tmanually using the '-m' option.)" << endl
    << "\t-r: recoverable mode, detect and restart failed node" << endl
      << "\t\t(Data Server failure detection always enabled.)" << endl
    << "\t-l: local execution, avoid rsh for running on same node" <<
endl
      << "\t\t(rsh no longer used for starting remote Data Servers.)"
<< endl;

   err_stream << instructions << endl;
   return;
}

//
// control verbose level for servers
//
extern int VERBOSE_LEVEL;

//
// flag to avoid rsh of starting a server
//
bool  avoid_rsh = false;

//
// local ip address, to be compared with each node ip to
// automatically determine whether to use rsh or sh
//
const char* curr_host;

//
```

Doc #541-0001442

NCR

Feature Unit Test Plan and Test Report

for

TOR DBMS Startup

January 6, 1999

Authors

Matt Dickey
Tak Sze

Data Warehouse Solutions

THIS DOCUMENT AND INFORMATION HEREIN ARE THE PROPERTY OF NCR CORPORATION AND ALL UNAUTHORIZED USE AND REPRODUCTION ARE PROHIBITED

*Copyright © 1998 by NCR Corporation, Dayton, Ohio U.S.A.
All Rights Reserved. Printed in U.S.A.
CONFIDENTIAL
Unpublished Property of NCR Corporation*

Revision History

Version	Author(s)	Principal Contributors	Date	Comments
A01	Matt Dickey		10/19/98	
	Tak Sze		11/17/98	
A02	Matt Dickey		1/6/1999	Include Unit Test Report
	Tak Sze			

Reviewers

Version	Name	Function	Organization
A01	Hoa Tran	Feature Team Leader	TOR Engineering
	Matt Dickey	Area Owner (s)	TOR Engineering
	Karen Kawate	Management Sponsor	TOR Engineering
	Mike Reed	Testware team member (s)	TOR Engineering

Approver

Version	Name	Function	Signature
A01	Karen Kawate	Management Sponsor	

Table of Contents

1 Introduction and Scope

This document covers the Unit Tests for the TOR DBMS Startup feature. This feature includes numerous changes to starter.cpp to support the use of NT services to perform multi-node startup of the database in an NT environment. In addition, six new executables have been created for the NT environment: torSvcConf, torStartSvc, torSrvSvc, chsvcconf, torsvclaunch and tordbabort. This document will also cover the tests for these programs.

torSvcConf is a program which installs or removes the NT service programs torStartSvc and torSrvSvc on single or multi-node systems. It is designed to be called from the TOR configuration utility and relies on the presence of the TOR configuration file tor_config.

The service program torStartSvc exec's the database startup program torstart and monitors it for unexpected terminations. The service program torSrvSvc is invoked by torstart once for each Data Server defined in the TOR configuration file. Each copy of torSrvSvc will exec a Data Server and monitor it for unexpected terminations.

The chsvcconf is a utility to allow the administrator to change the account and/password used by the TOR Services. The torsvclaunch is a utility used to start a specific TOR Service. The tordbabort program is used to abort the TOR DBMS when the normal TOR shutdown does not work.

This test plan will explain the various database configurations to be verified and the tests used to verify them. It will identify any limitations in the test coverage and all negative tests used to prove correctness of error handling.

1.1 Test Plan Schedule

	Start Date	End Date
Unit Test Plan	11/30/98	12/2/98
Unit Test Execution	12/3/98	12/18/98
Unit Test Report	12/18/98	12/18/98

1.2 Test Environment

Testing needs to be done in a number of differing environments in order to prove that the new startup procedures will work in single and multi-node environments, and that they are sufficiently robust to support configuration changes such as adding or removing Data Servers or SMP nodes.

HARDWARE
Single node testing will be done on a uniprocessor node with the following minimum characteristics:

- 266 MHz Pentium II processor
- 128MB RAM
- 1 4GB SCSI hard disk

Multi-node testing will be done on a system consisting of 4 SMP nodes, with each node having

- 4 400 MHz Pentium II processors
- 512MB RAM (or greater)
- 2 sharable SCSI RAID units (or more) with 20-40 4 or 9 GB SCSI drives
- TCP/IP connection through either Ethernet or Bynet

A number of tests will use only a subset of the nodes in this system; these tests will be identified as such in the Detailed Test Specification section.

SOFTWARE
Each node in the above hardware configurations should be running NT 4.0 with service pack 3. TOR software should be from IP2 or later.

TOR CONFIGURATIONS
The following list shows the various configurations to be explicitly verified as functional:

a) Configuration 1
- 1 Query Coordinator
  - 100 MB log
  - 200 MB disk device

a) Configuration 2:
- 1 Query Coordinator
  - 100 MB log
  - 50 MB disk catalog device
- 1 Data Server per node
  - 100 MB log
  - 200 MB disk devices

b) Configuration 3:
- 1 Query Coordinator
  - 100 MB log
  - 50 MB disk catalog device
  - 200 MB disk device
- 4 Data Servers per node
  - 100 MB log
  - 200 MB disk devices

For testing on the uniprocessor system, device sizes will be reduced by a factor of 5. Each of the 3 configurations given above will be tested on the uniprocessor system, and in 1, 2 and 4 SMP node versions of the large system.

1.3 Test Coverage Matrix

The test matrix groups the testable items according to feature areas within the startup subsystem. Each testable item is either classified as positive or negative test. A positive test is one testing the logic flow which assumes the system is behaving in a normal situation. A negative test is one testing the error flow that deals with abnormal situation such as running with a downed node or missing a configuration file.

Testable Item Identifier	Description	Test Category	Test Identifier
TOR Service Installation			
SRVINST1	Install TOR services on a single node system	positive	INSTALL1
SRVINST2	Install TOR services on a four node system	positive	INSTALL2
SRVINST3	Install TOR services with one data server	positive	INSTALL1
SRVINST4	Install TOR services with four data servers	positive	INSTALL2
SRVINST5	Invoke service install on the QC node	positive	INSTALL1, INSTALL2
SRVINST6	Invoke service install on a non-QC node	negative	INSTALL3
SRVINST7	Install with TORNODES registry entry missing	negative	INSTALL3
SRVINST8	Install with a TOR node not responding	negative	INSTALL3
SRVINST9	No service account entered on input	negative	INSTALL3
SRVINST10	No password entered on input	negative	INSTALL3
SRVINST11	No domain name specified on input	negative	INSTALL3
SRVINST12	Invalid domain name specified on input	negative	INSTALL3
TOR Service Removal			
SRVRMV1	Remove TOR services on a single node system	positive	INSTALL1
SRVRMV2	Remove TOR services on a four node system	positive	INSTALL2
SRVRMV3	Remove TOR services with one data server	positive	INSTALL1
SRVRMV4	Remove TOR services with four data servers	positive	INSTALL2
SRVRMV5	Invoke service removal on the QC node	positive	INSTALL1
SRVRMV6	Invoke service removal on a non-QC node	negative	INSTALL4
SRVRMV7	Remove with TORNODES registry entry missing	negative	INSTALL4
SRVRMV8	Remove with a TOR node not responding	negative	INSTALL4
SRVRMV9	Remove with a TOR Service registry entry missing	negative	INSTALL4
Starter and Server Services			
SERVICE1	Start the Starter Service	positive	INSTALL1, INSTALL2
SERVICE2	Start the Server Service	positive	INSTALL1, INSTALL2
SERVICE3	Abort the Starter Service	positive	SERVICE1
SERVICE4	Abort the Server Service	positive	SERVICE1
SERVICE5	Query the Starter Service	positive	SERVICE1
SERVICE6	Query the Server Service	positive	SERVICE1
SERVICE7	Get Starter Service status	positive	SERVICE1
SERVICE8	Get Server Service status	positive	SERVICE1
SERVICE9	Stop the Starter Service	negative	SERVICE1
SERVICE10	Stop the Server Service	negative	SERVICE1
SERVICE11	Send SCM control command > 128 to Starter Service	negative	SERVICE1
SERVICE12	Send SCM control command > 128 to Server Service	negative	SERVICE1
SERVICE13	The Starter process crashed	negative	SERVICE2
SERVICE14	The Data Server process crashed	negative	SERVICE2
SERVICE15	Starter executable is missing	negative	SERVICE2
SERVICE16	Data Server executable is missing	negative	SERVICE2
Starter			
STARTER1	Start TOR on a single node system	positive	START1
STARTER2	Shutdown TOR on a single node system	positive	START1

STARTER3	Start TOR on a four-node system	positive	START2
STARTER4	Shutdown TOR on a four-node system	positive	START2
STARTER5	Start TOR on a single Data Server system	positive	START3
STARTER6	Shutdown TOR on a single Data Server system	positive	START3
STARTER7	Start TOR on a four Data Server system	positive	START4
STARTER8	Shutdown TOR on a four Data Server System	positive	START4
STARTER9	Restart TOR on a single node system post-shutdown	positive	START1
STARTER10	Restart TOR on a four node system post-shutdown	positive	START2
STARTER11	Restart TOR on single Data Server system post shutdown	positive	START3
STARTER12	Restart TOR on a four Data Server system post shutdown	positive	START4
STARTER13	Automatic restart of a Data Server	positive	START3
STARTER14	Automatic DB shutdown after loss of QC	positive	START1
STARTER15	Automatic restart of Data Server on remote node	postivie	START2
STARTER16	Start TOR via Control Panel with arguments	positive	START2
STARTER17	Start TOR via torsc with arguments	positive	START3
STARTER18	Start TOR manually (without start service)	positive	START1
STARTER19	Start TOR from Config Utility	positive	START3
STARTER20	Use –m option to manually start a Data Server	positive	START8
STARTER21	Automatic restart of manually started server	positive	START8
STARTER22	Start without a valid TORHOME registry entry	negative	START9
STARTER23	Start without a valid TORNODES registry entry	negative	START9
STARTER24	Start without a valid CURRENTVERSION registry entry	negative	START9
STARTER25	Start with no TOR registry entries at all	negative	START9
STARTER26	Start with an invalid tor_config file	negative	START9
STARTER27	Start with no tor_config file	negative	START9
STARTER28	Start with a missing tor_port file	negative	START9
STARTER29	Start with 2nd node down	negative	START16
STARTER30	Start on 2nd node with QC node down	negative	START17
STARTER31	Start manually with invalid arguments	negative	START18
STARTER32	Start without Server Service installed	negative	START9
STARTER33	Torstart trace messages	positive	START20
STARTER34	Torstart display messages	positive	START21
Service Launch Utility			
LAUNCH1	Invoke TOR Start Service	positive	START3
LAUNCH2	Invoke TOR Server Service	positive	TORSC2
LAUNCH3	Invoke TOR Start Service from Config Utility	positive	START4
LAUNCH4	Invoke TOR Server Service from generated script	positive	START8
LAUNCH5	Invoke arbitrary (non-TOR) service	positive	TORSC5
LAUNCH6	Invoke TOR Start Service on remote node	positive	START8
LAUNCH7	Invoke TOR Server Service on remote node	positive	TORSC7
LAUNCH9	Invoke TOR Server Service on remote node from script	positive	TORSC9
LAUNCH10	Invoke arbitrary service on remote node	positive	TORSC10
LAUNCH11	Print torstart help via torsc	positive	TORSC11
LAUNCH12	Print torstart usage via torsc	positive	TORSC12
LAUNCH13	Verify correct argument passing from torsc to torstart	positive	TORSC2
LAUNCH14	Verify correct argument passing from torsc to DS	positive	TORSC14
LAUNCH15	Invoke TOR Start Service without service installed	negative	TORSC15
LAUNCH16	Invoke TOR Server Service without service installed	negative	TORSC15
LAUNCH17	Invoke TOR Server Service on already running DS	negative	TORSC17
LAUNCH18	Invoke service on non-connected node	negative	START17
LAUNCH19	Invoke service while missing TOR rgeistry entries	negative	TORSC19
LAUNCH20	Verify incorrect arguments passed to torstart	negative	START18

Abort Utility			
ABORT1	Abort TOR services on a single node system	positive	ABTUTIL1
ABORT2	Abort TOR services on a four node system	positive	ABTUTIL2
ABORT3	Abort TOR services with one date server	positive	ABTUTIL1
ABORT4	Abort TOR services with four data servers	positive	ABTUTIL2
ABORT5	Invoke abort utility on the QC node	positive	ABTUTIL1
ABORT6	Invoke abort utility on a non-QC node	positive	ABTUTIL2
ABORT7	Abort with some services already stopped	negative	ABTUTIL2
ABORT8	Abort with some service registry entries removed	negative	ABTUTIL2
ABORT9	Abort with TORNODES registry entry removed	negative	ABTUTIL2
ABORT10	Abort with some "TOR Node Service" registry entries removed	negative	ABTUTIL2
Change Service Account Utility			
CHSVC1	Change service account name	positive	CHANGE1
CHSVC2	Change service account password	positive	CHANGE1
CHSVC3	Change service account domain	positive	CHANGE1
CHSVC4	Change service account on the QC node	positive	CHANGE1, CHANGE2
CHSVC5	Change service account on a non-QC node	positive	CHANGE2
CHSVC6	Change service account on a single node system	positive	CHANGE1
CHSVC7	Change service account on a four node system	positive	CHANGE2
CHSVC8	No password specified on input	negative	CHANGE1
CHSVC9	No account name specified on input	negative	CHANGE1
CHSVC10	No domain name specified on input	negative	CHANGE1
CHSVC11	Invalid domain name specified on input	negative	CHANGE1
CHSVC12	Invoke change with a node not responding	negative	CHANGE2
CHSVC13	Invoke change with some services already stopped	negative	CHANGE2
CHSVC14	Invoke change with some service registry entries removed	negative	CHANGE2
CHSVC15	Invoke change with TORNODES registry entry removed	negative	CHANGE2
CHSVC16	Invoke change with some "TOR Node Service" registry entries removed	negative	CHANGE2

1.4 Exceptions and Test Limitations

There are no known limitations to the tests.

1.5 Detailed Test Specification

This section provides a detailed description of each test, which testable items the test is testing, how each test is to be performed, and the expected result.

Reference test suites. Test suite location may be specified in lieu of defining each test.

1.5.1 Test INSTALL1

Testable items covered: SRVINST1, SRVINST3, SRVINST5, SRVRMV1, SRVRMV3, SRVRMV5, SERVICE1, SERVICE2

1. Configure a 4-node TOR DBMS with four Data Servers on each node, and install the TOR Services via the configuration utility. Use a local user account for the TOR Services. Do not start the TOR Services. Run the Change Service Account utility, chsvcconf.exe on a QC node. Change the TOR Service account to a domain account. Start the Starter.
2. Invoke the utility on a non-QC node. Change the TOR Service account password to an invalid password. Shutdown TOR and restart the TOR Services.
3. Shutdown one of the non-QC nodes. Invoke chsvcconf.exe on the QC node. After the change is made, bring up the downed node.
4. Remove the TORNODES registry entry on the QC node. Invoke chsvcconf.exe on the QC node. Restore the TORNODES registry entry after the change is made.
5. Remove one of the "TOR Node Service" registry entries on the QC node. Invoke chsvcconf.exe on the QC node. Restore the "TOR Node Service" registry entry after the change is made. Restart the TOR Services.

Pass/Fail Criteria:
The following are the expected results for each steps in the test description:
1. All TOR Services should be changed with the new account and can be started.
2. None of the TOR Services can be started due to invalid password.
3. The change should fail due to one of node being downed.
4. The change should fail due to the TORNODES registry entry missing.
5. The change should succeed. All TOR Services can be started.

1.6 Dependencies

The execution of this unit test plan depends on the TOR configuration utility provided by Brian Hamilton.

2 Test Summary

This section provides a summary of the tests run, the number of tests, and any outstanding major issues.

Number of tests run	32
Number of tests passed	32
Number of tests failed	0
Number of tests re-run	7

The only major problem currently outstanding is that restart of Data Server processes fails in server.exe. The process is correctly restarted by torstart.exe, but the Query Coordinator process does not correctly register the new Data Server process. See Problem 1 in the Summary of Problems section. The other outstanding problem is that sometimes a shutdown command hangs in the Query Coordinator process. This problem happens only infrequently and has a simple workaround (just running tordbabort will complete the shutdown).

2.1 Test Conclusions

- The features tested are ready for use in an NT environment.
- The features have not been regression tested in the Solaris environment.

2.2 Test Coverage Matrix

No direct tests were run on a four-node system as part of the unit test. Testing was instead limited to one or two node systems, with from 1 to 4 Data Servers per node. Testing outside this test plan has run successfully on a four-node system.

Two tests were added to the set of tests for torstart.exe. These tests (START21 and START22) explicitly test the interfaces to the error logging and tracing facility to verify correct functioning of the trace retrofit feature for torstart.exe.

2.3 Test Results

2.3.1 Summary of Problems

This section summarizes major problems or categories of problems found. All of the problems listed here were in torstart.exe, torsc.exe, torsvcconf.exe, or torSrvSvc.exe, with the exception of 1 and 2, where the problem appears to be in server.exe.

1. Restart of a failed data server does not work completely from a system perspective because the Query coordinator does not properly register the new Data Server. The code in torstart.exe which handles the restart, however, appears to work perfectly. Also had to add call from torstart.exe to register the new server, which works OK. This problem affected tests START2, START3 and START8

2. Shutdown from pdsh sometimes fails to work if the Query Coordinator does not stop, causing torstart.exe to wait forever in waitForShutdown(). This problem has to lie in the server.

3. Shutdowns also failed due to errors in waitForShutdown(), where '=' was used where '==' was intended. Fixed in source. Also had to fix torSrvSvc code to get correct server exit code and return appropriate enum value back to torstart so that monitor() can correctly distinguish aborts, shutdowns, and crashes. All fixed in source.

4. File propagation had to be fixed to handle case where no files are found to be propagated since NT FindFirstFile() and FindNextFile() return different error codes if no files match the pattern. The NT documentation fails to mention what FindFirstFile() will do if no first file is found. Also, in our code, the wrong file name was getting built because strcpy() was mistakenly used instead of strcat(). Fixed in source.

5. Argument processing in torsc was flaky due to an improper initialization value for the argument count in getCLopt(), also the serviceName was getting improperly checked by getCLopt() and should not have been placed in the argument list. Fixed in source.

6. Torsc blew up when given the command ' torsc TORsrvr9' because it attempted to access arguments that did not exist. Fixed in source by putting in checks for existence of arguments before accessing them.

7. Xcopy of manual startup script to a remote node does not work, apparently due to coalescence of double backslashes one too many times.

8. Received NT error 1060 if we tried to start services without the appropriate administrative privileges. Fix by using UserManager to provide proper 'Log on as a Service' privilege. This requirement is documented in the IP4D release notes.

9. Tor_port file was being accessed without the proper path prefixed to the file name. Fixed in source.

10. If torsvcconf.exe is run on a node which does not have tor_config, the utility crashes due to config_info not checking for file existence. torsvcconf.cpp is modified to check for the existence of tor_config before proceeding to install the services. Fixed in source.

11. Event logging and tracing did not work due to problems in pieventdataaccess.cpp, which ignored the final .eda file included in the event_en.eda file. The startup code's message file was the final file. Fixed in source for smlayer_fc/pieventdataaccess.cpp.

12. Manual startup script created by torstart.exe had incorrect service name for the Query Coordinator. Fixed by modifying script creation code to use correct service name as set up by torsvcconf.exe.

13. Event logging files did not set up properly due to improper initializations in perl scripts. Fixed by a change to eventfmt.pl.

2.3.2 Problems Resolved

This section lists and briefly describes all major problems found which have since been fixed, and provide the resolution to the problems.

Information in this section is presented as a table. Problem tracking # refers to the number of the problem in the list given in previous section of this document.

Table 1: Summary of Major Problems Resolved

Problem Tracking #	Problem Description	Resolution
3	WaitForShutdown() failed to detect server shutdowns, causing hangs of torstart.	a) Fixed conditionals which were doing assigns instead of equality comparisons. b) Also fixed code in torSrvSvc to return correct exit code, which was sometimes mangled due to a prematurely closed handle.

4	File propagation failed.	a) Fixed construction of file names, where strcpy() was mistakenly used insteadof strcat(). b) Had to check for a different return code from NT system call to detect case where no files existed to be propagated.
5	Torsc sometimes failed to send correct arguments through to service.	a) Torsc was incorrectly passing the service name as argv[0]. b) Argument counter in getCLopt() was off by one after removing the service name. c) Some improper arguments could not be detected due to use of an uninitialized variable.
6	Torsc attempted to check non-existent arguments, causing exception.	a) Fixed argument checking loops to check argument count before attempting to dereference argv[] pointers.
7	Copy of manual startup .bat file to remote node using xcopy fails due to loss of backslashes in pathnames.	a) Recreate pathnames with extra backslashes.
8	Unexpected NT 1060 error.	a) Add section to release notes on how to properly set up privileges for using and administrating services.
9	The tor_port file was not getting accessed correctly.	a) Fixed the pathname setup to have the proper directory.
10	Torsvcconf.exe could blow up if the tor_config file was not present.	a) Put a check into the code which handles the configuration file to make sure the file exists before attempting to read it.
11	Final .eda file included in event_en.eda by mergeeda.pl was inaccessible at runtime.	a) Fixed loop end conditionals in smlayer_fc/pieventdataaccess.cpp to be '<=' instead of '<'.
12	Manual startup script used incorrect service name.	a) Fixed script creation code to correspond with proper service name as set up by torsvcconf.exe.
13	Multiple .eda files did not set up correctly.	a) Fixed indexing in loop in eventfmt.pl to correctly initialize whole array instead of just first element.

2.3.3 Problems Outstanding

This section lists and briefly describes all problems found which are unresolved at the time of this report. The tracking number refers to the problem numbers used in section 2.3.1, while the '#' column cross-references items in the Detailed Test Results section.

Table 2: Summary of Major Problems Outstanding

Problem Tracking #	#	Plan for Resolution

1 TAR 96984	8,9,11	Create TAR for server.exe with explanation of where problem shows up in the code in which the Query Coordinator processes registrations from Data Servers.
2 TAR 97045	8	Verify conditions under which the 'shutdown' command fails to stop the Query Coordinator and create a TAR describing problem.

2.3.4 Detailed Test Results

The Problem tracking # column refers to the problem numbers used in the 'Summary of Problems' section. The tests marked as P/I passed as far as the targeted test feature is concerned, but encountered problems in other areas which caused the test to be incomplete insofar as strict adherence to the test specification is required.

Table 3: Test Results

#	Test ID	Start Date	End Date	Test Status	# New Problems Found	# Regress-ions Found	# Problems Left	Re-run #	Problem tracking #'s (or comments)
1	INSTALL1	12/3	12/3	P	0	0	0	0	
2	INSTALL2	12/3	12/3	P	0	0	0	0	
3	INSTALL3	12/4	12/4	P	0	0	0	0	
4	INSTALL4	12/7	12/8	P	1	0	0	0	
5	SERVICE1	12/14	12/14	P	0	0	0	0	
6	SERVICE2	12/15	12/15	P	0	0	0	0	
7	START1	12/10	12/10	P	0	0	0	0	
8	START2	12/14	12/14	P/I	2	0	2	1	1, 2, 3, 4
9	START3	12/14	12/14	P/I	0	0	0	0	1
10	START4	12/14	12/14	P	1	0	0	1	10
11	START8	12/14	12/14	P/I	5	0	0	5	1, 5, 7, 8, 9
12	START9	12/15	12/15	P	0	0	0	0	
13	START16	12/15	12/15	P	0	0	0	0	
14	START17	12/15	12/15	P	0	0	0	0	
15	START18	12/15	12/15	P	0	0	0	0	
16	START20	12/18	12/21	P	1	0	0	1	11
17	START21	12/18	12/21	P	1	0	0	1	13
18	TORSC2	12/16	12/16	P	0	0	0	0	
19	TORSC5	12/16	12/16	P	0	0	0	0	
20	TORSC7	12/16	12/16	P	0	0	0	0	
21	TORSC9	12/16	12/16	P	0	0	0	0	
22	TORSC10	12/16	12/16	P	0	0	0	0	
23	TORSC11	12/16	12/16	P	0	0	0	0	
24	TORSC12	12/16	12/16	P	0	0	0	0	(update UTP)
25	TORSC14	12/16	12/16	P	0	0	0	0	
26	TORSC15	12/16	12/16	P	1	0	0	1	6

27	TORSC17	12/16	12/16	P	1	0	0	1	12
28	TORSC19	12/16	12/16	P	0	0	0	0	
29	ABTUTIL1	12/8	12/8	P	0	0	0	0	
30	ABTUTIL2	12/9	12/9	P	0	0	0	0	
31	CHANGE1	12/10	12/10	P	0	0	0	0	
32	CHANGE2	12/11	12/11	P	0	0	0	0	

2.4 Test Deviations and Test Limitations

The tests START2, START3 and START8 are limited (and marked as incomplete in the detailed test results table) because of a problem in the Query Coordinator which causes the QC to abort when it encounters a restarted Data Server (problem #1 form the Summary of Problems section). Since torstart.exe's Data Server restart code functions correctly, the tests were deemed to have passed nonetheless.

2.5 Tools

Most of these tests are not candidates for automation since they require frequent manual intervention (killing Data Servers, downing nodes, etc.) and some navigation through GUI's. Some simple positive tests can be automated by preparing batch scripts which perform the steps given in the test (and possibly assuming that the installation and/or configuration steps have been previously performed).

2.6 Glossary

QC	Query Coordinator. There is only one Query Coordinator running on each TOR DBMS. The QC is the server which is responsible for the TOR DBMS catalog.

2.7 References

1. High-Level Design Specification for TOR 2.0, Doc. # 541-0001110A02 (or latest revision)

2. Product External Specification for TOR 2.0, Doc. # 541-0001111A04 (or latest revision)